UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/607,593	06/27/2003	Diaa Fathalla	MSFT125572	6013

38991 7590 01/12/2007

CHRISTENSEN, O'CONNOR, JOHNSON, KINDNESS, PLLC
1420 FIFTH AVENUE
SUITE 2800
SEATTLE, WA 98101-2347

EXAMINER	
VU, TUAN A	
ART UNIT	**PAPER NUMBER**
2193	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	01/12/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No.	Applicant(s)	
	10/607,593	FATHALLA, DIAA	
	Examiner	Art Unit	
	Tuan A. Vu	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *06 November 2006*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-26* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-26* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 11/06/2003.

As indicated in Applicant's response, claims 1-10, 19-23 have been amended. Claims 1-26 are pending in the office action.

Specification

2. The disclosure is objected to because of the following informalities: the disclosure at different places mentions about a 'API call replay tool' and a 'API replay tool'; and since there is no consistency as to when this above tool is a just replay tool as opposed to a call replay tool, absent any explicit definition to each concept, there is no clear way to construe this tool as being one or the other. A tool for replaying an *API call* in common-accepted semantic cannot be equivalent to a API replay tool; because a *API call* is very distinct from just a API in the same manner as a call to a function (i.e. dynamically contextual) is different from just a function (i.e. statically declarative). In order for such tool limitation to have consistent and patentable meaning, there must be an extensive correction to propagate this concept throughout the Disclosure so that it is expressed by an unequivocal and consistent terminology. The above inconsistency in terminology will be treated as though the replay tool is to put together a API for a contextual instance.

Appropriate correction is required.

Claim Objections

3. Claim 1 is objected to because of the following informalities: The term 'APT call code' (line 7) looks like a typo error of what is recited as API call code. Appropriate correction is required.

4. Claims 1, 11, and 19 recite 'symbol table ... for mapping references within ... record into a memory space ... replay tool'; and 'creating ... code sequence ... corresponding to ... call record; wherein memory references within ... code sequence are specified according to a set of mapping entries within the symbol table'. The references perceived as being within the call record are recited as being mapped into a memory space of a tool; and there appears to be a non-connection between this memory space allocation with the subsequent creation of code sequence using the same call record, such that when the memory sequences of the API code sequence are specified based on a mapping set of entries (which appear to come from nowhere) within the symbol table, there is no use of the memory space respective to the earlier symbol table mapping implication. That is, it is observed that the language describing the use of symbol table in conjunction with the memory references for a API call in light of a replay tool memory space is not accurate or inefficient for laying out a flow of action that interrelate claimed elements which would lend to a clear understanding of what the application is all about. The Disclosure depicts reading a log and generating a API builder with support from a symbol table such that the builder creates API constructs using call back knowledge and memory manager. The disclosure as a whole does not provide a corresponding teaching that would match or exactly justify the use of the language as observed in the claim in light of the above un-correlated limitations as set forth above. The claim language would have to be corrected to establish a more fluent flow of actions that meaningfully inter-correlate the constituents of each respective separate action.

Correction is required lest this would lead to a USC 112 type of rejection (Emphasis added)

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

> The changes made to 35 U.S.C. 102(e) by the American Inventors Protection Act of 1999 (AIPA) and the Intellectual Property and High Technology Technical Amendments Act of 2002 do not apply when the reference is a U.S. patent resulting directly or indirectly from an international application filed before November 29, 2000. Therefore, the prior art date of the reference is determined under 35 U.S.C. 102(e) prior to the amendment by the AIPA (pre-AIPA 35 U.S.C. 102(e)).

6. Claims 1-9, 11-17, 19-25 are rejected under 35 U.S.C. 102(e) as being anticipated by Ringseth et al., USPN: 6,625,804 (hereinafter Ringseth).

> The applied reference has a common assignee with the instant application. Based upon the earlier effective U.S. filing date of the reference, it constitutes prior art under 35 U.S.C. 102(e). This rejection under 35 U.S.C. 102(e) might be overcome either by a showing under 37 CFR 1.132 that any invention disclosed but not claimed in the reference was derived from the inventor of this application and is thus not the invention "by another," or by an appropriate showing under 37 CFR 1.131.

As per claim 1, Ringseth discloses a computer device containing an application program interface (API) replay tool for creating and submitting API calls based upon input API call records, the tool comprising:

a symbol table for mapping references (e.g. symbol table 430 – Fig. 4) within an input API call record (e.g. parse tree, Converter 424 – Fig. 4; *UEPM source* - Fig. 5; Fig. 7-8) into a memory space allocated to the API call replay tool (e.g. Fig. 3; Table 1- col. 8; event framework – Fig. 3; col. 6, line 40 to col. 7, line 60; Table 2 – col. 9; *state 434, Attribute provider 470* – Fig. 4; Fig. 7, 8a); and

an API call builder for creating an API call code sequence (e.g. *_interface* -Fig. 10A-B) for invoking an API call corresponding to the input API call record (e.g. step 524→528; step 570 --Fig. 5; Fig. 6), wherein memory references within the API call code sequence are specified according to the mapping entries within the symbol table (e.g. step 528 – Fig. 5; col. 11, lines 49-57; col. 12, line 41 to col. 13, line12)

As per claim 2, Ringseth discloses managing memory block per context of call records (namespace – Fig. 7, 8a; namespace N – Fig. 10b; *source events, receiving events* – Table 1, Table 2 – col. 8, col. 9)

As per claim 3, Ringseth discloses firing of asynchronous event and handlers as needed, resulting in managing of events as received(or as thread identified) as in multi or single thread implementation (see col.7, lines 11-45)

As per claim 4, Ringseth discloses execution template for maintaining set of resources associated with input API call record (e.g. col. 12, lines 60-65; col. 14; *template* – col. 14, lines 17-26; state 434 – Fig. 4; col. 22, *ATL template code* – Note: state information from compiler parsing and establishing events in a information state to be used by the Attribute provider reads on set of resources; and template ATL reads on set of resources associated with events being parsed)

As per claim 5, Ringseth discloses a API call executioner showing code sequence and set of resources (e.g. Fig. 4-5; Fig 6a-6b)

As per claim 6, Ringseth discloses event handlers within the tool to render a API call (see - Fig. 17, 18a-b, 19; *delegate ... event handlers* - in related text)

As per claim 7, Ringseth discloses handlers providing callback (e.g. *callback* – col. 6, lines 45-52; Fig. 15c; Fig. 8a; col. 23, lines 43 to col. 24, line 28 – Note: unhooking reads on callback functions; see Fig. 19)

As per claims 8-9, Ringseth discloses references as pointers and variables (col. 8, lines 41-56; Attribute provider – Fig. 4; events, allocation for variables – col. 12, lines 10-47; col. 15, *V-Function Pointer-Based Eventing protocol implementation*; Fig. 7).

As per claim 11, Ringseth discloses a method for replaying API calls comprising

mapping references (within a API call record … space allocated to an API call replay tool);

creating API call code sequence … memory references are specified … mapping entries); all of which limitations being included in the subject matter claimed in claim 1. Hence, the claim is rejected using the corresponding rejection as set forth therein.

As per claims 12-17, these claims correspond to claims 2-5, 8-9, respectively; hence are rejected using the corresponding rationale as set forth therein, respectively.

As per claim 19, this claim is a computer-readable medium version of claim 11 and includes the same subject matter recited therein; hence is rejected using the corresponding rejection as set forth therein.

As per claims 20-25, these claims correspond to claims 2-5, 8-9, respectively; hence are rejected using the corresponding rationale as set forth therein, respectively.

7. Claims 1, 10-11, 18-19, and 26 are rejected under 35 U.S.C. 102(e) as being anticipated by Chong et al., USPubN: 2003/0208743 (hereinafter Chong).

As per claim 1, Chong discloses a computer device containing an application program interface (API) call replay tool for creating and submitting API calls based upon input API call records, the API replay tool comprising:

a symbol table for mapping references (e.g. Fig. 11) within an input API call record (Workflow input file – Fig. 6b; Input file – Fig. 11; para 0113, pg. 8) into a memory space allocated to the API call replay tool (e.g. Model analyzer 114 – Fig. 11; Workflow - Fig. 12, Fig. 6b; internal model 206– Fig. 8; Fig. 9; para 0193-0195, pg. 13); and

an API call builder for creating an API call code sequence (e.g. para 0055, pg. 4; Actions – Fig. 12; *action ...new object that implements ... interfaces* – para 0197, pg. 13; para 0200, 0212, 0221, 0255, 0290) for invoking an API call corresponding to the input API call record (e.g. Workflow input – Fig. 6B; para 0158, pg. 11), wherein memory references within the API call code sequence are specified according to the mapping entries within the symbol table (e.g. Fig. 11, Fig. 17).

As per claim 10, Chong discloses passing assembly language instructions (para 0038 – pg. 3).

As per claim 11, Chong discloses a method for replaying API calls comprising

mapping references (within a API call record ... space allocated to an API call replay tool);

creating API call code sequence ... memory references are specified ... mapping entries); all of which limitations being included in the subject matter claimed in claim 1. Hence, the claim is rejected using the corresponding rejection as set forth therein.

As per claim 18, refer to rejection of claim 10.

As per claim 19, this claim is a computer-readable medium version of claim 11 and includes the same subject matter recited therein; hence is rejected using the corresponding rejection as set forth therein.

As per claim 26, refer to claim 18.

Response to Arguments

8. Applicant's arguments filed 11/06/06 have been fully considered but they are not persuasive. Following are the Examiner's observations in regard thereto.

Rejections under 35 USC 102(e) using Ringseth:

(A) The Applicant has submitted that Ringseth does not disclose 'a symbol table for mapping references within an input API call record ... memory space ... for creating a API call code sequence' (Appl. Rmrks, pg. 12, middle) wherein 'memory references ... are specified according to a set of mapping entries ... symbol table' as recited in claim 1(Appl. Rmrks, pg. 13, bottom); and this on several points.

First, it is argued that Ringseth's symbol table is for type information used for the UEPM constructs, and such table is not for mapping references within an input API record into a memory space allocated to the replay tool (Appl. Rmrks, pg. 14, top half). In reply, it is noted that the action of mapping references from a record into a memory space of a tool signifies that some references from a record has been mapped into the tool as the tool is being used to put together a API call implementation or instance thereof, all of which according to broad reasonable interpretation of the so-recited *mapping references* by way of a *symbol table*. Fundamentally a symbol table is known to have declared variables or function references statically derived from the source code, and the atomic elements from this table will be referred

to in order to support compiling actions such as establishing calling graph leading to resolution at link time. For one skill in the art, the linking of different code calling references (e.g. attribute, parameters, class, function characteristics, access type, or even alias) is based on the referencing requirements being laid out from the compiler-generated control flow graph or interference tree which support or identify the memory references needed to effectuate call time (invocation) of different functions, those that are originally founded using the basic elements of the symbol table or source input. Moreover, a tool to set up the code assembling/linking is always imparted with a proper memory space in order for the references needed therefor to be properly allocated a valid address/value based on the requirements of the code call graph (see Ringseth: *Error checker, converter and parse tree* – Fig. 4), i.e. for each such requirement from the compiler-established such graph – or call record, a reference mapping to a proper value of memory is fundamentally required. The scenario whereby content of the source code as static symbol tokens --established in a symbol table-- leads to generating for all call references (i.e. a mapping from a call requirement – as a call graph for example --into a appropriate resolved address location during a linker resolution) a corresponding call/runtime time valid instance of memory space/value allocated for such reference requirement **amounts to** what the claimed limitation is all about. That is, symbol table for mapping references within a call record (call record being the instance of call or constructs required for a code sequence – as in a call tree) into a space allocated (resolved address location for any such reference) to the replay tool (memory space for the tool to establish mapping references to its valid address).

The claim (claim 1) does not make it clear as to what exactly such *API call record* is all about; nor does it make it sufficiently explicit *how the memory references being specified*

according to a set of mapping entries (a very convoluted phraseology) *within the symbol table* is all about; nor does the claim really teach what exactly the act of *mapping references within a... record* into a space is all about; nor does it describe what *API replay tool space* amounts to; or what exactly 'creating an API call code sequence for invoking an API call corresponding to ... a record' entails (invoking suggesting a non-static call).

An API call record is treated as a structure being established prior to the sequence of API code targeted --by the replay tool or code builder -- to be put together, because lacking any timeframe as to the history of this record, the record amounts to a either a set of pre-stored code template/constructs or a structure relating code calling hierarchy as a tree or a flow graph that happens to exist prior to the invoking of the API call.

If a call record is established and includes memory references then the rationale behind of the invoking (which entails a dynamic runtime action) of a API call ('creating an API call code sequence for invoking an API call corresponding to ... a record') become unclear when 2 unclear limitations happen to obfuscate any rationale for understanding the above *invoking* process: *set of mapping entries within the symbol table* are recited along with *for mapping references within an input API call record.* Symbol table and a static record cannot be explicitly used for runtime unless the invention provides specific teaching as to how this (dynamic) invoking is done using static data. It is unclear how the mapping is effected (is it API call references being mapped from table entries?) in light of indefinite teachings conveyed as memory references being *specified* based of *a set of mapping entries* (loaded phraseology without real support to its surface) *within a symbol table* along with what is recited as *symbol table for mapping references within an input API call record.* One skill in the art would not be able to construe the existence

of memory references being specified from the following: (i) a *set of mapping entries* (no clear teaching about what mapping entry entails) within a symbol table are based upon to specify memory references within a API call sequence whereas this sequence is for dynamically invoking a call based on some static record; (ii) the symbol table is for mapping references within a API call record into a API call memory space, (iii) such memory space unspoken of during the course implementing the API call invoking. Apparently at least two distinct actions are using elements of each other but are clearly not heading towards a common purpose, but rather set forth to accomplish disparate results independent of each other. That is, the mapping of references (via the symbol table) into a memory space does NOT appear to remotely relate to the memory references specification for an targeted API call sequence when such references are based on a set of mapping entries (of unknown origin) within the symbol table.

Third, memory references being specified according to a set of entries in a symbol table amount to a very vague rationale as to what useful or viable action this is all about. How a symbol table – symbol with a full and well-accepted meaning of it -- be construed as supporting a mapping of recorded references (call record, i.e. how can a call record which contain contextual runtime elements relate to rather and static non-contextual symbols is unclear) into a space so that such space be allocated to possibly contain memory references of a API call sequences which are in turn founded on memory references based on a unknown *set of mapping entries* within the very same symbol table is not clear. Established compiler technologies would have a symbol table to provide atomic static elements such that these would be needed to establish logical constructs as in sequence of interference (of flow relationship) code constructs which lead to establishing of appropriate memory allocation of each of code construct instance

(variable, pointer, function, parameter, constant, alias) involved in such interrelationship required to effect a proper linking process purported for a call to be proper and non-faulty; as set forth above in the previous section.

The rejection using Ringseth has set forth use of symbol table to establish a interrelationship of code constructs via some tree utilized by a converter and memory reference mapping based attribute provider to provide back end code implementation for an event type of application interface. And this is consistent with the interpretation of the claim language in view of the above observations. That reads on the mapping based on using a symbol table support for creation of API call sequences wherein the memory references are derived from a record --which is founded on symbol table contents; the relationship between the API required memory references to the call record analogous to a mapping; and the correspondence between elements of the symbol table and the requirements derived from the API record (as in a tree) analogous to another mapping. Applicants' argument that the recited limitations are not found in Ringseth amount to assertions founded on other connotations that are different from broad and reasonable interpretation currently stemmed from the claim language, in view of its very flaws as set forth above.

Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

(B) Second, Applicant has disagreed that '_interface' as proffered by the Office action cannot in any way amount to a sequence of API code sequence (Appl. Rmrks, pg. 14 bottom). It is noted that API code sequence merely amounts to a sequence of code constructs that pertain to an

application interface; and Ringseth by providing object oriented intermediate code checking so to effectuate code sequence in C++ to address event handling does provide code sequence for a application interface. For one skill in the art, the mere code to address event is intrinsically a interface especially the event is founded on COM objects as purported by Ringseth. The argument is not persuasive.

(C) Third, Applicant has presented Ringseth flow of steps taken to make use of the intermediate code and transform the parse tree in order to assert that Ringseth does not teach memory references according to the mapping entries within the symbol table (Appl. Rmrk, pg. 15). The argument amount to an biased understanding that cannot be reasonably and broadly construed from reading the claim; and the deficiency of the language used in the claim to describe the rationale of the claimed invention has been set forth in length from above. Ringseth has provided mapping and use of symbol table according to a call record and references required therein; i.e. the argument will be referred back to section A.

(D) Applicant has submitted that Ringseth fails to discloses claims 11-17 in terms of fulfilling 'mapping references' and 'creating API code' and 'memory references ... are specified ... mapping entries' (Appl. Rmrks, pg. 16). However, claim 11 recites the limitations of claim 1; the argument will be referred back to section A.

(E) Applicant has submitted that Ringseth fails to discloses claim 19 in terms of fulfilling 'mapping references' and 'creating API code' and 'memory references ... are specified ... mapping entries' (Appl. Rmrks, pg. 16). The argument will be referred back to section A.

Rejections under 35 USC 102(e) using Chong:

(F) Applicant has submitted that Chong's displaying a view for deferencing and garbage collection is not same as 'mapping references' using a symbol table and a API call record as claimed (Appl. Rmrks, pg. 18, bottom, pg. 19, top). Chong teaches obtaining parsed content from source files and using symbol table according to which to support the model analyzer to create code emitting via workflow analysis including variables mapping based on a view instance. The record has been analogized to the source input wherein references of code required via parsing are established and translated into a model view, based on which checking process and variable mapping are effected to generate interface code to implement object-oriented objects thus viewed from the model analyzer. The paradigm reads on the recited *mapping*, a memory space of the viewer, symbol table and *memory references* within a *record* to *invoke a API call sequence*; and Chong's paradigm is not far different from Ringseth use of symbol table to support establishing of runtime required interrelation between code constructs to effect proper building of code for which references are resolved via the tree parsing or the model checking. All the observations made against the deficiency of the claim language (i.e. disparate elements recited but uncorrelated in terms of a meaningful rationale in view of well-established concepts) are applied herein (refer to Claims Objections, and section A above); and the Applicant's arguments against Chong are non-persuasive in view of the previous sections. The argument against garbage collection does not even take into the slightest consideration how the rejection has mapped the limitations against the cited portions. Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

(G) The arguments against Chong with respect to claims 11, 18 , 19 and 26 (Appl. Rmrks, pg. 19-20) will be referred back to section F in light of the observations in section A.

The claims will stand rejected as set forth in the office Action.

Conclusion

9. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (272) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Meng-Ai An can be reached on (571)272-3756.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before

using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Tuan A Vu
Patent Examiner,
Art Unit 2193
January 9, 2006